Algorithmic Exploration in Graph Theory

Celina Hwang, Lena Lee, Emma Liu
(Mentors: Tomasz Slusarczyk, Julia Kozak)

May 2025

Abstract

This paper explores the fundamentals of graph theory and how they relate to the foundation of algorithms, which are essential in today's interconnected world. Specifically, it looks into the Greedy Algorithm's use in graph coloring and explores algorithms for generating spanning trees. In order to address these algorithms, the paper dives into concepts of connectivity, cycles, chromatic numbers, trees, and spanning trees, which show how graph theory is applied to real-life examples like airline routes. This exploration underscores the importance of graph theory in understanding advanced modern computation systems.

1 Introduction

When one mentions graphs, most people often think of Cartesian graphs on the xy-plane with slopes and y-intercepts. However, these are not the only types of graphs that exist: in graph theory, a graph is a set of vertices and edges. These graphs appear in GPS systems, social media connections, and even in airlines. Though we don't realize it so much, graphs are becoming more prevalent in this highly interconnected world. They are the key to the development of new technologies regarding connectedness, and most important of all, algorithms. An algorithm is a set of rules or processes followed to solve a problem or for a certain performance. The graphs we discuss in this paper are of these kinds: GPS, social media, and airlines. Graph theory, which discusses the underlying rules of graphs and the numerous ways in which graphs can be utilized, has become one of the most important topics in understanding the internet and its vast connectivity.

This paper explores graph theory, focusing on its connectivity and how it relates to the formation of algorithms.

In Section 2, we present the basics of graph theory and define essential terms, such as connectivity, neighborhood, and isomorphism, which are crucial in explaining the algorithms in later sections.

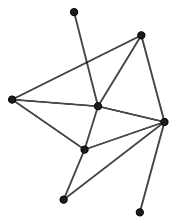
Section 3 discusses the concept of trees, which is an expanded concept from the connectivity of graphs. This section demonstrates how graph theory provides the foundation for many algorithms and highlights its significance.

Lastly, Section 4 discusses coloring and chromatic numbers, which are significant topics that connect graph theory with algorithms and are commonly used in real life, such as in airline planning.

2 Graph Theory

In this section, we introduce foundational graph theory definitions.

Definition 2.1 (Graph)**.** A *graph* is a pair (V, E) where V is the vertex set and E is the edge set. *Vertices* are points on a graph, and *edges* are lines connecting two distinct vertices. An edge e in edge set E can be described as a pair $\{x, y\}$, where edge e connects two distinct vertices, x and y, from the vertex set V. In graph theory, $\{x, y\}$ can be denoted as xy for convenience.



Definition 2.2 (Isomorphic)**.** Two graphs are considered *isomorphic* to each other if they preserve the same number of vertices and edges, as well as all edge connections. In graph theory, even if two isomorphic graphs do not appear to be structurally identical, they are considered the same graphs.

Definition 2.3 (Planar Graphs)**.** A *planar graph* is a graph where none of its edges "overlap" or intersect at points other than shared vertices.

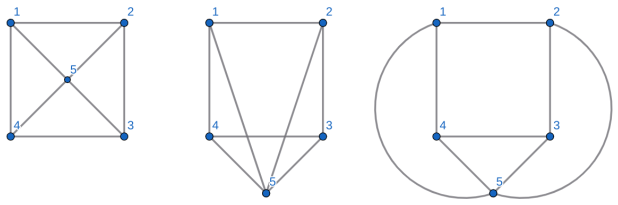


Figure 1: These three graphs are all isomorphic to each other, as they all have the same vertex and edge sets. The rightmost graph is a planar isomorphism of the leftmost graph, which is also planar.

Definition 2.4 (Neighbor)**.** Two vertices x, y in graph G are *neighbors* if they are adjacent and are connected by an edge. In other words, x and y are neighbors if $xy \in E$. The set of all neighbors of a vertex v is called its *neighborhood*, and is denoted as $N(v)$.

Definition 2.5 (Degree)**.** The degree of a vertex v can be defined as the number of neighbors vertex v holds. This is denoted as $\deg(v)$, and can be defined as $\deg(v) = |N(v)|$, where $|N(v)|$ is the size of the neighborhood of v.

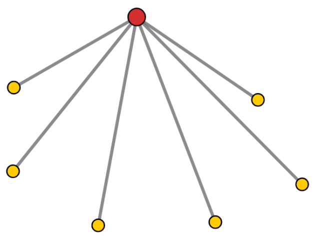


Figure 2: The yellow vertices are neighbors of the red vertex, and the degree of the red vertex is 6.

Definition 2.6 (Path)**.** A graph P with vertex set $V = \{v_0, v_1, ...v_k\}$ and edge set $E = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k\}$.

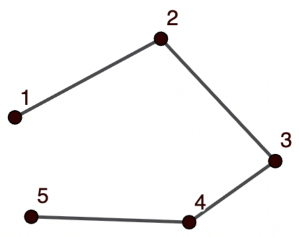


Figure 3: A path exists from vertex 1 to vertex 5.

Definition 2.7 (Cycle)**.** A graph K with vertex set $V = \{v_0, v_1, ...v_k\}$ and edge set $E = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k, v_kv_0\}$. In other words, a path with an additional edge v_kv_0 added so that it starts and ends at the same vertex. Cycles can be denoted as C^n for n-cycle for a cycle of length n.

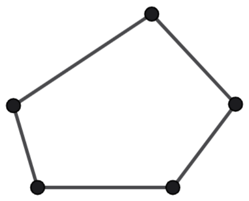


Figure 4: Example of a 5-cycle.

Definition 2.8 (Connectivity)**.** A graph G is *connected* if there exists a path between every pair of distinct vertices. Likewise, a graph is *disconnected* if there is no path between every pair of vertices.

Definition 2.9 (Complete Graph)**.** A *complete graph* is a graph where there exists an edge between any two vertices. It can also be called a *clique*, and is denoted as K^n for a clique of size n, where n is the number of vertices in the graph.

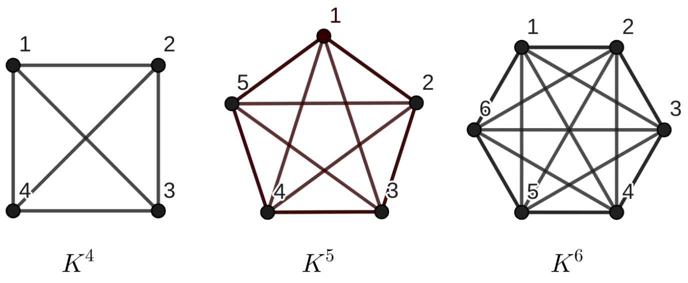


Figure 5: Examples of cliques.

Definition 2.10 (Subgraph)**.** A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$. Thus, we can say that $G' \subseteq G$.

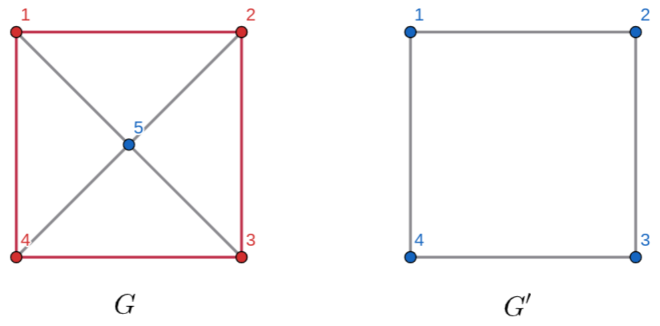


Figure 6: G' is a subgraph of G, as it contains a subset of G's vertices and edges (highlighted in red on G).

Another useful lemma in graph theory is the *Handshaking Lemma*, which gets its name from how we calculate the number of total handshakes exchanged in a room, given that we know exactly how many people each person shook hands with.

Lemma 2.1 (Handshaking Lemma)**.** The number of edges in a graph G equals half of the sum of the total degrees of all vertices in G:

$$\sum_{v \in V} d(v) = 2|E(G)|$$

Proof. By adding the total degrees of every vertex in a graph, each edge is counted twice, once for each end of the edge. Therefore, in order to correct the double-counting we must divide this sum by 2, which results in the total number of edges in the graph. □

3 Trees

Definition 3.1 (Tree)**.** A *tree* is a connected graph that is acyclic (not containing any cycles). There are 3 defining features of trees, two of which we will prove later in this section.

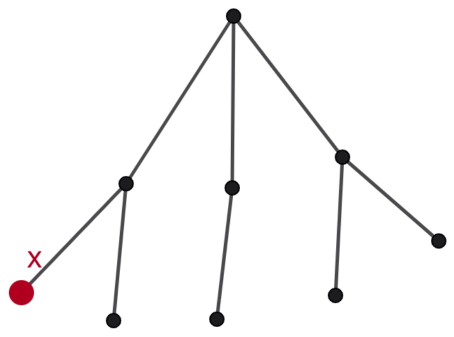


Figure 7: Example of a tree.

Definition 3.2 (Leaf)**.** A *leaf* is a vertex on a tree that has degree 1. Every tree has at least 1 leaf. In Figure 7, vertex x is a leaf.

Definition 3.3 (Bridge)**.** *Bridges* are edges on a connected graph G such that removing them would cause G to become disconnected. In other words, bridges are edges that are necessary to form certain paths between pairs of vertices.

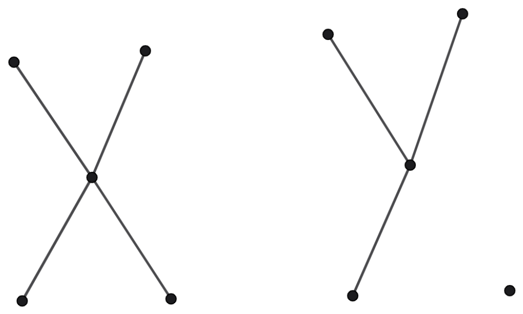


Figure 8: A connected graph and the resulting disconnected graph when a bridge is removed

Definition 3.4 (Spanning Tree)**.** A spanning tree T of a connected graph G is a subgraph of G such that T is a tree and contains all of G's vertices (V(T) = V(G)). We will prove that every connected graph has a spanning tree later in this section.

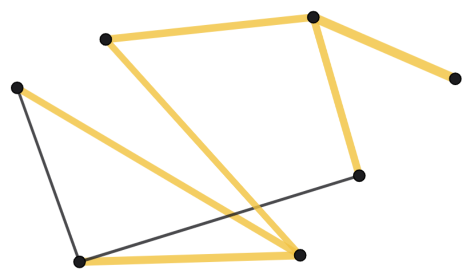


Figure 9: An example of a spanning tree on a connected graph (highlighted in yellow).

Lemma 3.1 (Trees)**.** Trees have 3 defining features:

1. All trees are connected graphs with n vertices and have $n - 1$ edges.
2. Trees contain no cycles.
3. There is a unique path between any two vertices on a tree.

Proof. We will prove that statement 3 implies statement 2 using proof by contrapositive, where we will show that if 2 is not true, then 3 is not true. This can be utilized because a true statement (i.e. $P \to Q$ is true) will result in a true contrapositive $\neg P \to \neg Q$ is true. Let us assume that statement 2 is not true and there is a graph with at least one cycle. Consider two vertices x and y on the graph. If the graph has at least one cycle, there are at least two paths that can be taken to travel between two vertices x and y around the cycle. □

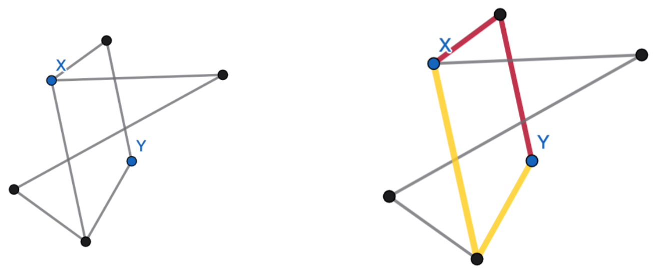


Figure 10: Vertices x and y on the graph have at least two paths between them.

Thus, we can conclude that if a graph is not acyclic, there must be more than 1 unique path from vertex x to vertex y. By proving that the contrapositive of the statement that 2 implies 3 is true, we have also proven the original statement.

We will also prove that every tree with n vertices has $n - 1$ edges using induction:

Proof. In our base case, where $n = 1$, there is only one vertex and exactly 0 edges, which satisfies the statement that there are $n - 1$ edges

In the inductive step, we assume that any n-vertex tree has $n - 1$ edges and prove that an $(n+1)-$vertex tree T has n edges. If we let v be a leaf on T and remove v to create the graph T' $(T' = T \setminus V)$. T' remains connected and acyclic because v is a leaf and does not lie on any path between two vertices x and y where $x, y \neq v$. Thus, we know that T' is an n-vertex tree with $n - 1$ edges, so $|E(T)| = n$. □

Lemma 3.2. Removing a leaf from a tree will still result in a tree.

Proof. Because removing a leaf removes a vertex and an edge, the relationship between the number of vertices n and the number of edges $|E|$ $(|E| = n - 1)$ still remains the same. Removing a leaf also maintains the graph's acyclic nature because it does not connect any two vertices on the graph. Finally, the edge between the leaf and its neighbor is not required to preserve the graph's connectivity, as it is only needed for paths between other vertices and the vertex that has been removed. Thus, removing the leaf will still result in a graph that's a tree. □

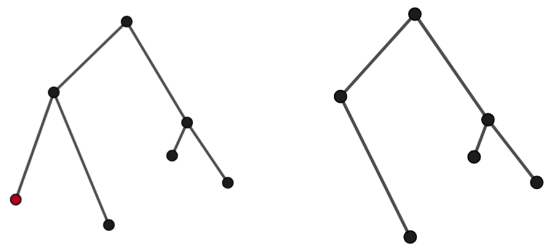


Figure 11: A tree before and after a leaf is removed.

Lemma 3.3. Trees consist entirely of bridges

Proof. Since there is only one unique path between every pair of vertices in a tree, each edge is part of at least one path between vertices. Removing an edge from a tree would break the path between some pair of vertices and disrupt the tree's connectivity. Thus, we can conclude that all edges in a tree are bridges. □

We are now going to look into two algorithmic methods used to generate spanning trees:

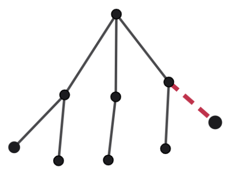


Figure 12: Removing a bridge results in two disconnected components.

Lemma 3.4 (Spanning Tree Algorithm 1)**.** A spanning tree can be generated from a connected graph by adding edges without creating cycles. Let $G = (V, E)$ be a connected graph with n vertices.

1. Set F equal to $\emptyset$.
2. Set U equal to $\{u\}$, where u is a vertex in $V(G)$.
3. While there is a vertex v of V such that v is not in U, add e from E to F such that e is not already in F and e adds v to U. The terminal graph $T = (U, F)$ is a spanning tree of G.

Proof. Let G be a connected graph with n vertices. We will let the spanning tree T on G have an empty edge set F and the vertex set consist of a single vertex u. We can now construct the spanning tree using an algorithm that iterates over a set of steps, adding an edge for every $n-1$ vertices that are not yet on the graph. This ensures that we do not create any cycles by visiting a vertex already in the vertex set. We can also think of this as ”visiting” every vertex that is not yet connected until all vertices have been added to the vertex set. Since we add an edge for every $n-1$ vertices, there are $n-1$ edges for a total of n vertices, and our resulting graph T is a spanning tree. $\square$

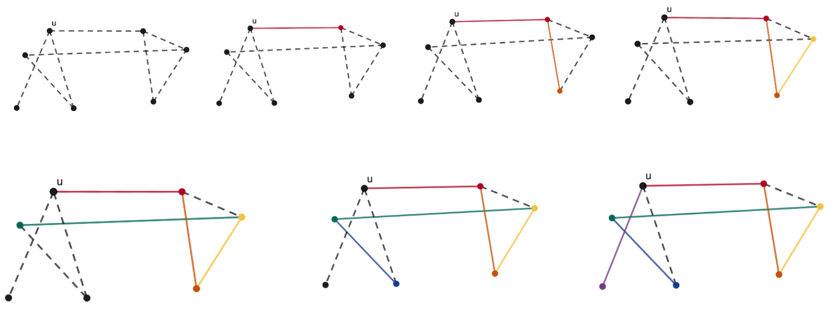


Figure 13: A spanning tree on a connected graph with an initially empty set of edges (denoted by dashed edges) created from adding edges (outlined in color).

Lemma 3.5 (Spanning Tree Algorithm 2)**.** A spanning tree can be generated from a connected graph by removing edges without disconnecting the graph. Let $G = (V, E)$ be a connected graph with n vertices.

1. Set F equal to E
2. While there is an edge α of F such that α is not a bridge of the graph $T = (V, F)$, remove α from F. The terminal graph $T = (V, F$ is a spanning tree of G.

Proof. When we remove some edge from G such that it is not a bridge, we remove an edge of the tree that is not required to keep the graph connected. In other words, we cannot remove any edges that would prevent the graph from having a unique path between a pair of vertices. In the process, G will stay connected and preserve all of its original vertices, and any existing cycles will be broken because only one unique path will remain between two vertices. We know that our subgraph with no bridges also must be a tree because trees consist entirely of bridges, as we already proved in lemma 3.3. Thus, after removing all non-bridge edges, the remaining subgraph will be a spanning tree. □

An example of the process of using this algorithm to generate a spanning tree on a connected graph is illustrated below:

The Spanning Tree algorithms mimics those used in computer science in the sense that they iterate over a series of repeated operations, just as algorithms go through a sequence of instructions to perform a computation. Similarly, the algorithms are repeated until a certain condition is met, which echoes the function of a while loop in programming.

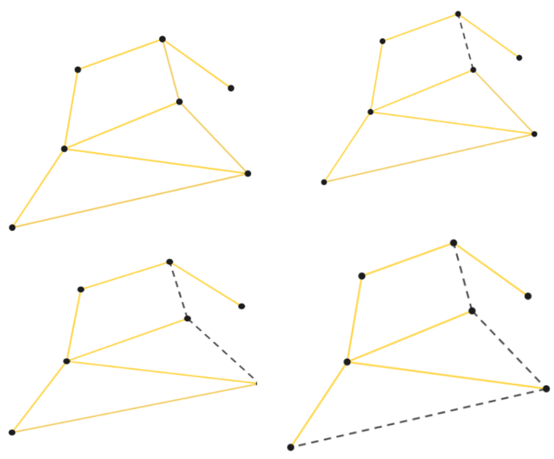


Figure 14: Non-bridges are removed from the connected graph (denoted by the dashed edges) to form a spanning tree.

4 Coloring and Chromatic Number

Definition 4.1 (Vertex coloring)**.** *Vertex coloring* is the process of assigning colors to the vertices of a graph G such that no two adjacent vertices are the same color.

Definition 4.2 (Chromatic Number)**.** $\chi(G)$ is the minimum number of colors required to do a vertex coloring of graph G.

Lemma 4.1 (Even Cycles)**.** Consider a cycle C^n, where n is even. Label all the vertices as $v_1, v_2, ...v_n$, and assign v_1 with color 1. Since v_2 is adjacent to v_1, we must assign it to the next color, color 2. However, v_3 can be color 1 again because it is not neighbors with v_1. Repeating this process, we can color all odd vertices with color 1 and all even vertices with color 2. In this way, no two neighboring vertices are the same color, and we can accomplish the vertex coloring of an even-length cycle with just two colors, therefore $\chi(G) = 2$.

Lemma 4.2 (Odd Cycles)**.** If we use the same method as in the previous lemma, then we run into a problem: v_n and v_1 are neighbors and are both odd. If we assigned odd vertices to one color and even vertices to another, then these two adjacent vertices would be the same color! Since this is not allowed, we must assign v_n to a new color 3, making $\chi(G) = 3$ for any odd cycle.

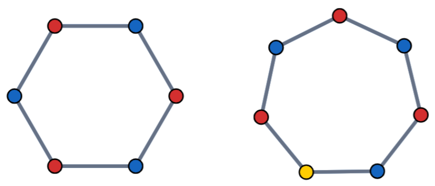


Figure 15: Examples of colored even and odd cycles.

Definition 4.3 (Maximum Degree)**.** ΔG is the maximum degree of any vertex in graph G.

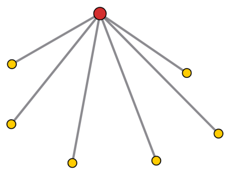


Figure 16: $\Delta G = 6$, as the red vertex has the highest degree (6) in this graph, while all other yellow vertices have degree 1.

The *Greedy Algorithm* is an algorithm that can be used to determine an upper bound for the chromatic number of any graph. We consider a fixed vertex enumeration $v_1, ...v_n$ of G and consider each vertex in the sequence, coloring each v_i with the first available color that has not already been used to color a neighbor of v_i in $v_1, ...v_{i-1}$ This ensures that the number of colors used never exceeds $\Delta G + 1$. However, using this algorithm produces a very large upper bound, so the chromatic number is usually smaller than this. Indeed, the upper bound is typically only applicable for complete or odd-cycle graphs.

Theorem 4.1.

$$\Delta(G) + 1 \geq \chi(G)$$

Proof. Suppose a vertex v is chosen from graph G, and $d = \Delta(G)$, or the maximum degree of G. Then any v has at most d neighbors, and in the worst case scenario all its neighbors have to be different colors. Therefore, we must assign v to another color distinct from the d colors already used, thus the upper bound on $\chi(G)$ is $d + 1$, which is equivalent to $\Delta(G) + 1$. □

Lemma 4.3. The vertices of any tree T can be colored black and white so that neighboring vertices receive different colors. In other words, $\chi(T) = 2$

Proof. We can start at the root of the tree and color it black first. Then we must color all its neighbors white, as they cannot be the same color. Now we can color the white vertices' neighbors black again, since we know that the root and its neighbors' neighbors cannot be neighbors again as trees are acyclic. Repeating this process, we can color the entire tree with just two colors! Therefore, we have proved that the chromatic number of any tree is 2. □

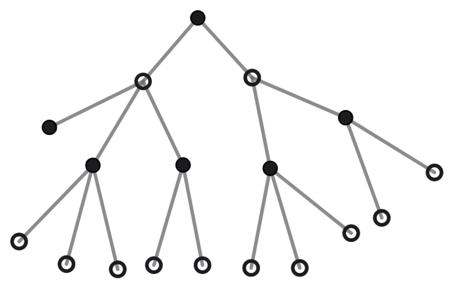


Figure 17: A tree where all its vertices are colored with two colors (black and white).

Theorem 4.2 (4-Color Theorem)**.** The *4-color theorem* states that if graph G is planar—we can draw the graph without intersecting edges—then G can be colored with at most 4 colors where no adjacent vertices share the same color. In other words, $\chi(G) \leq 4$. The 4-color theorem had been a popular problem for mathematicians that was unsolved until 1976, when Kenneth Appel first proved the theorem with computer assistance. It was also one of the first theorems that could only be proven with extensive computer calculations.

5 Application Problem

Problem 5.1 (Airline Problem)**.** Suppose we only have two airlines: a red airline and a blue airline. The red airline offers flights to different locations on a tree, and the blue airline offers flights on any odd cycle. You want to go on a very long vacation, so you decide to take a round trip by using an odd number of flights to start and end your journey in the same location. However, the blue airline is more expensive, so we want to find a way to minimize the number of blue flights we take. Show that we only need one blue flight to travel on an odd cycle.

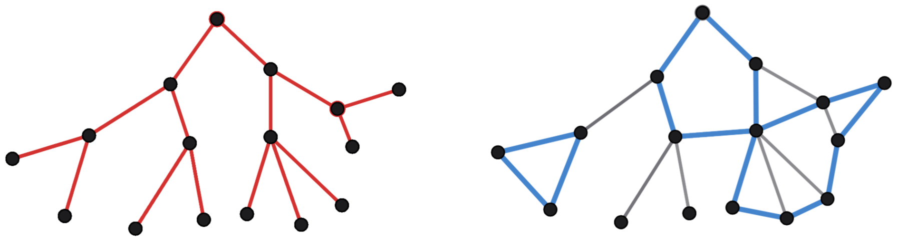


Figure 18: The tree on the left shows a possible configuration for the red flights, while the right shows examples of odd cycles.

Proof. Let the number of flights needed be n. Because we are traveling on an odd cycle, the number of flights can be considered the set of edges, which must mean that we also land at n locations (or the set of vertices).

Because trees are acyclic, it is not possible to travel on any cycle on our red airline tree. Therefore, we know that we must have at least one blue flight that is NOT included in the tree.

We can approach this problem by using Lemma 4.3, which proves that the chromatic number of a tree is 2. Coloring the red airline tree in black and white, we notice that every edge in the tree has different-colored endpoint vertices. This must be true of every edge in this tree by the very definition of vertex coloring, where adjacent vertices cannot be the same color. Therefore, an edge represents a red flight if and only if it has one white and one black vertex. By our problem statement, we have to travel on an odd cycle, meaning that we have an odd number of vertices. However, if we try to color an odd cycle using just black and white, by Lemma 4.2 there will always be one pair of adjacent vertices that are the same color, like in Figure 19. Therefore, we know that this one flight cannot be on the red airline tree, and therefore it must be a blue flight. □

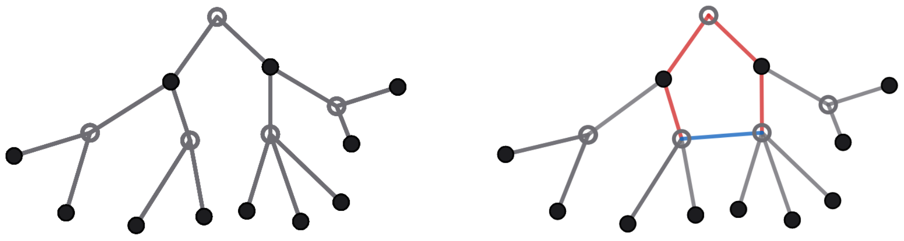


Figure 19: One possible 5-flight trip including 4 red flights and one blue flight.

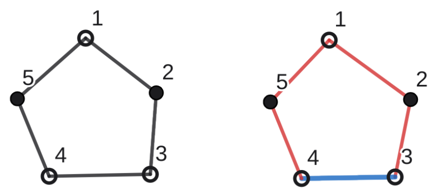


Figure 20: Vertices 3 and 4 are both white, thus the edge 34 must represent a blue flight.